Administrivia

- Co-Instructor Introduction
- HW1 in progress, due next Wednesday.
  - Primers on various topics posted on the class webpage.
- Quizzes each week, starting next week. 1 week to complete. Any score > 50% counts for full points.
- TA introduction slides posted on the class webpage.
- Slides posted after the class.
- TA Office Hour schedule coming soon.
  - Mine will be Friday mornings at 9.15-10.15 a.m. each week
- Some movement on add/drop, some of you added. Prioritizing by date of graduation, and when you came on the waitlist. Speak with me if you have an extraordinary need to take the class.

Lecture 2: Linear Regression (Part 1)

CIS 4190/5190

Spring 2023

Recap: Types of Machine Learning

- **Supervised learning**
  - **Input:** Examples of inputs and desired outputs
  - **Output:** Model that predicts output given a new input

- **Unsupervised learning**
  - **Input:** Examples of some data (no “outputs”)
  - **Output:** Representation of structure in the data

- **Reinforcement learning**
  - **Input:** Sequence of interactions with an environment
  - **Output:** Policy that performs a desired task

Recap: The Machine Learning Pipeline

Data Z

Machine learning algorithm

Model f

Recap: The Machine Learning Pipeline

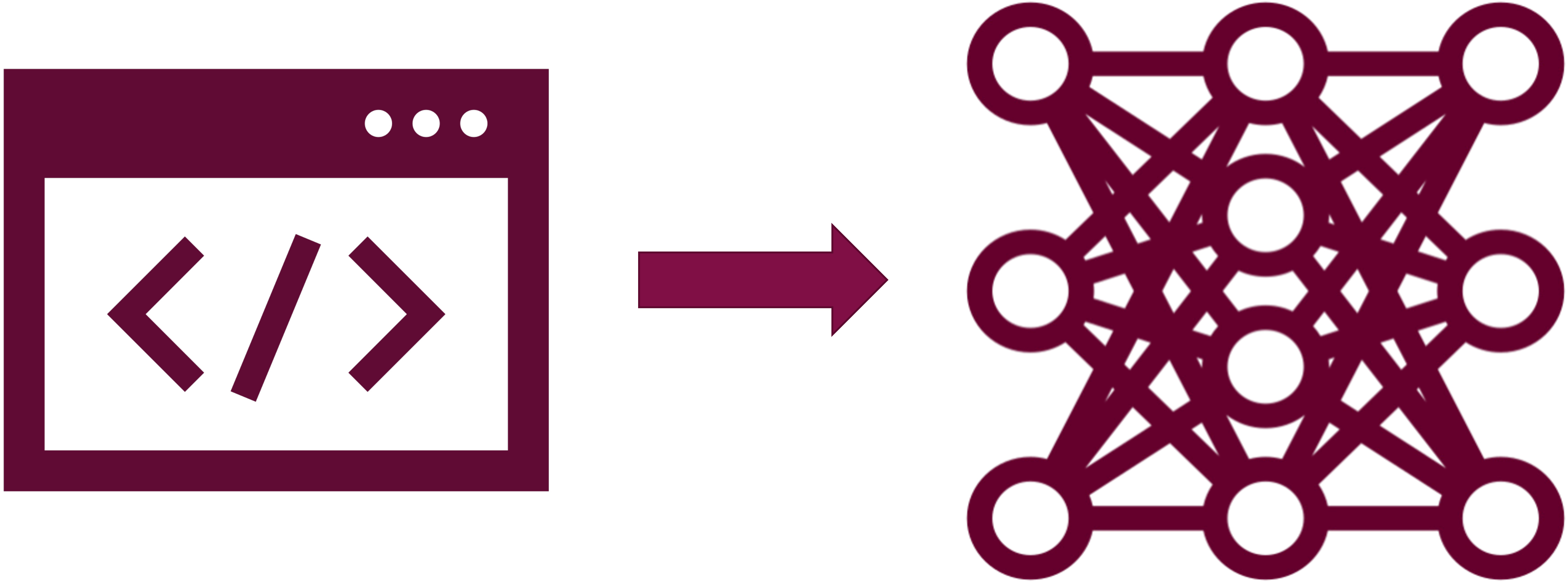
Think of this learned model as replacing a manually written function in code

```
output = function(input)
```

Supervised ML as Programming 2.0

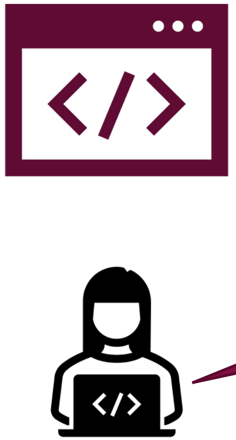
Traditional Programming

Machine learning (ML)

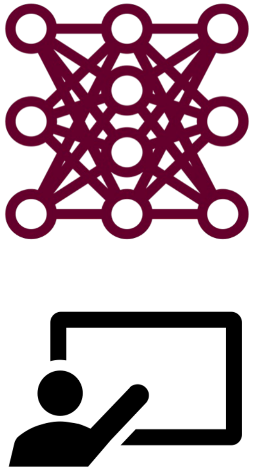


The key difference lies in how the “programmer” specifies tasks to the computer

Supervised ML task specification: ~~programs~~ examples



Here is a program to implement Newton's second law of motion



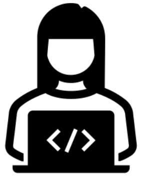
Here are some examples. Try to imitate them.

```python
def compute_force(m, a):
    '''
    returns force (in N) needed to
    move mass m (in kg) at
    acceleration a (in m/s^2)
    '''

    F = m * a
```

Mass m (kg)	Acceleration a (m/s^2)	Force F (N)
2.5	4	10
5	2	10
20	0.5	10
40	0.25	10
40	2.5	100
20	5	100

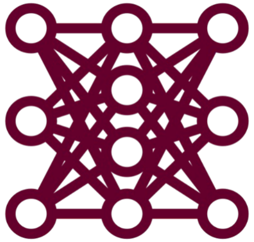
It seems a bit silly to teach Newton's law by examples, when you can code it up …

Supervised ML task specification: ~~programs~~ examples



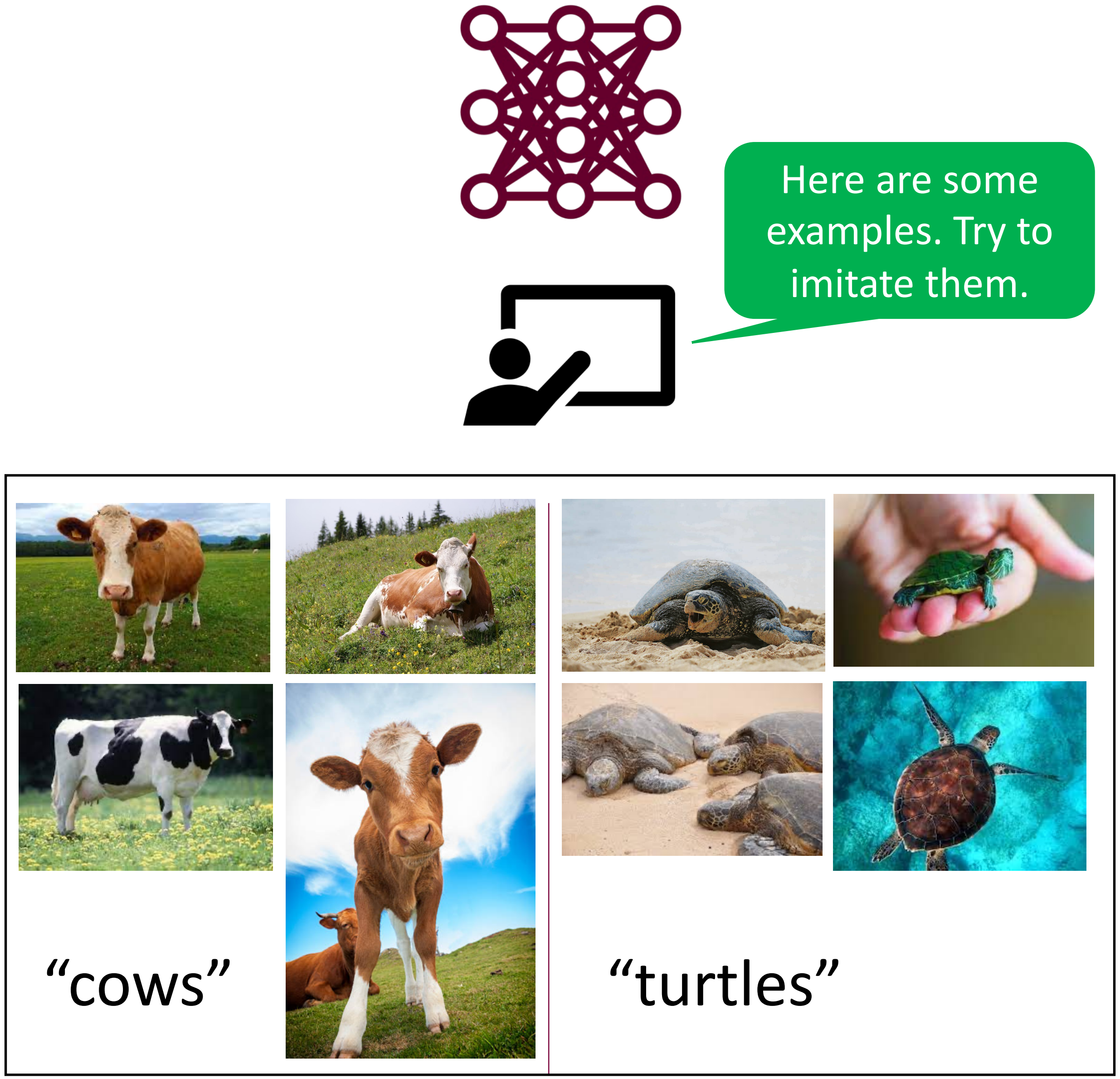
Here is a program to recognize an image as a cow or a turtle

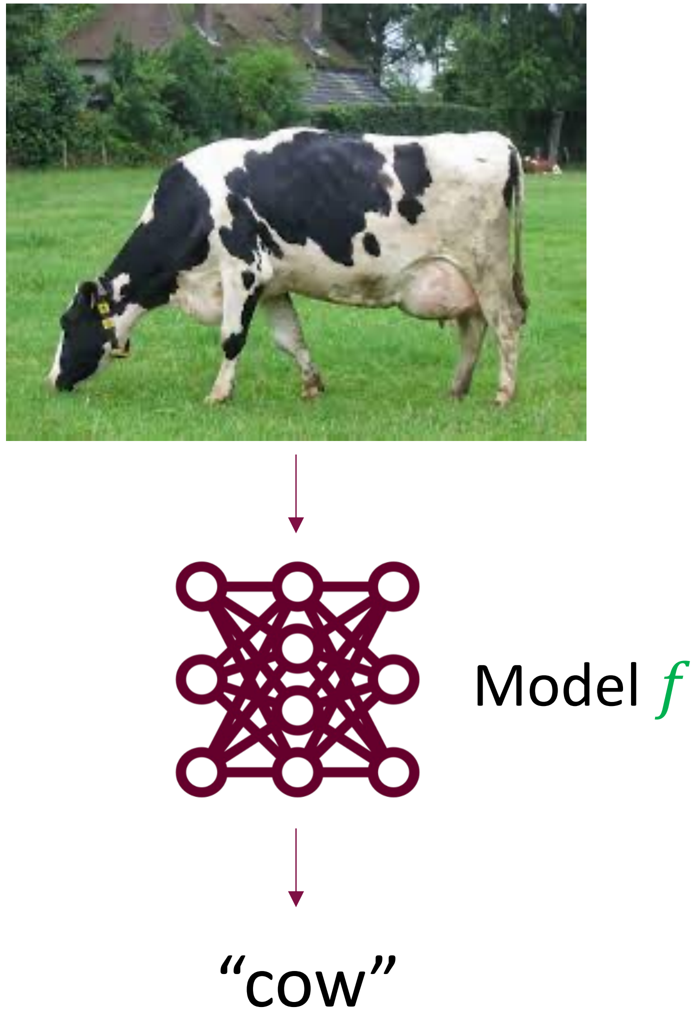
```
def cow_or_turtle(image):

    ???
```



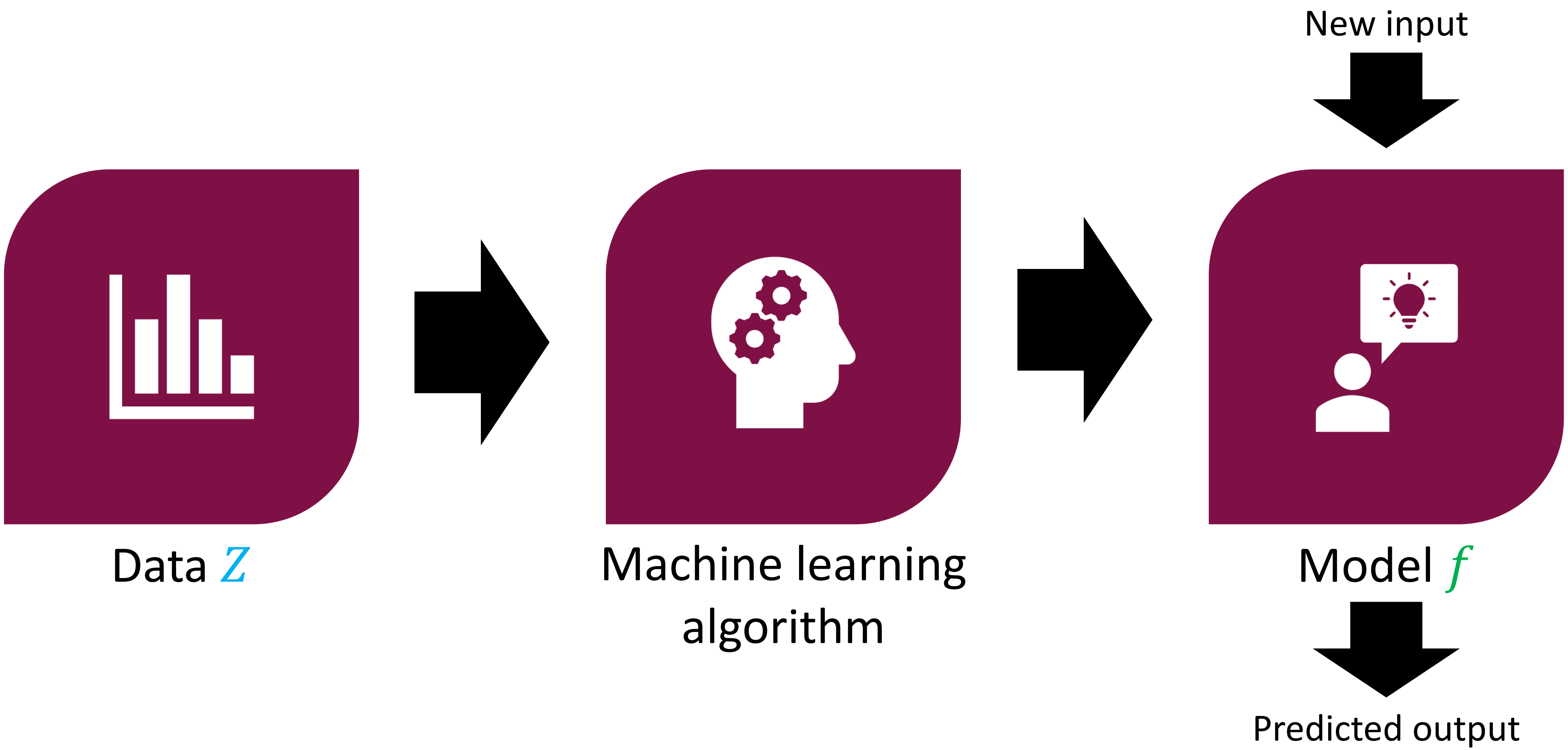
Here are some examples. Try to imitate them.

Putting the trained ML system to use





Designing the ML pipeline: The Hypothesis Class



Design Choice:

What **model family** (a.k.a. **hypothesis class**) to consider when looking for f?

Linear Functions

- Consider the space of linear functions $f_\beta(x)$ defined by

$$f_\beta(x) = \beta^\top x$$

Linear Functions

- Consider the space of linear functions $f_\beta(x)$ defined by

$$f_\beta(x) = \beta^\top x = [\beta_1 \quad \cdots \quad \beta_d] \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} = \beta_1 x_1 + \cdots + \beta_d x_d$$

- $x \in \mathbb{R}^d$ is called an **input** (a.k.a. **features** or **covariates**)
- $\beta \in \mathbb{R}^d$ is called the **parameters** (a.k.a. **parameter vector**)
- $\hat{y} = f_\beta(x)$ is called the **output** (a.k.a. **predicted label**)

Linear Regression Problem

"Target labels", or just "labels"

- **Input:** Dataset $Z = \{(x_1, \boxed{y_1}), \ldots, (x_n, \boxed{y_n})\}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$
- **Desired Output:** A linear function $f_\beta(x) = \beta^\top x$ such that $y_i \approx \beta^\top x_i$

- **Typical notation**
  - Use i to index examples (x_i, y_i) in data Z
  - Use j to index components x_j of $x \in \mathbb{R}^d$
  - x_{ij} is component j of input example i

- **Goal:** Estimate $\beta \in \mathbb{R}^d$

Linear Regression Problem

- **Input:** Data $Z = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$
- **Output:** A linear function $f_\beta(x) = \beta^\top x$ such that $y_i \approx \beta^\top x_i$


Photo by NASA Goddard

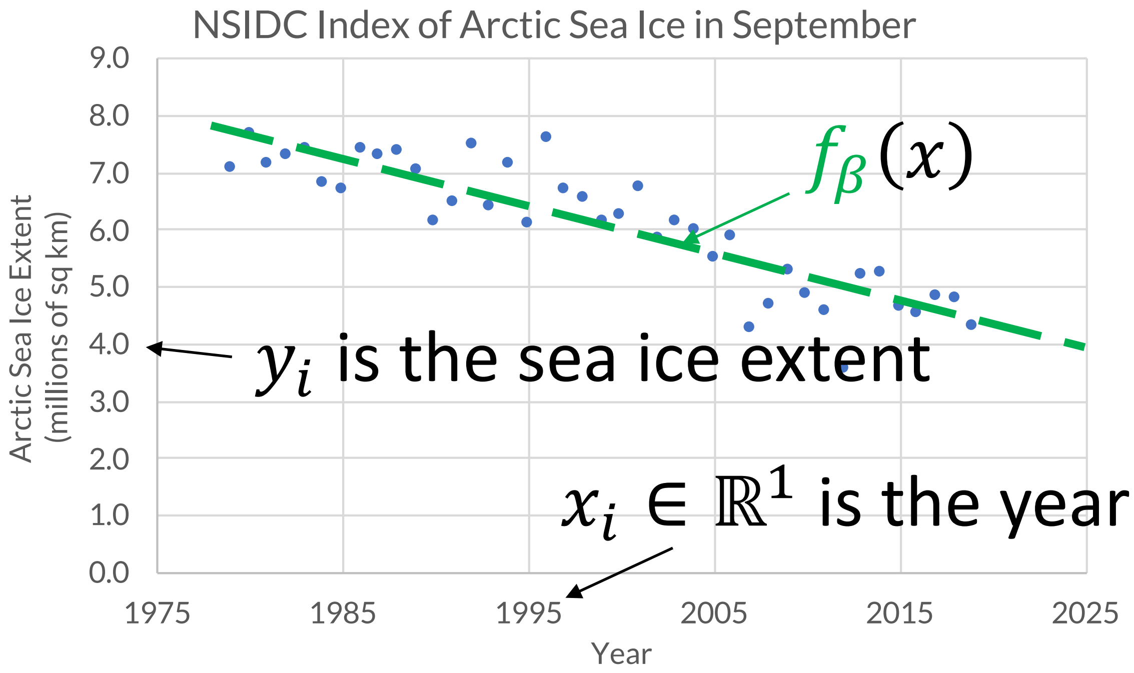


Image: https://www.flickr.com/photos/gsfc/5937599688/
Data from https://nsidc.org/arcticseaicenews/sea-ice-tools/

Linear Regression Problem

Design Choice: What does this mean?

- **Input:** Data $Z = \{(x_1, y_1), \dots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$
- **Output:** A linear function $f_\beta(x) = \beta^\top x$ such that $y_i \approx \beta^\top x_i$

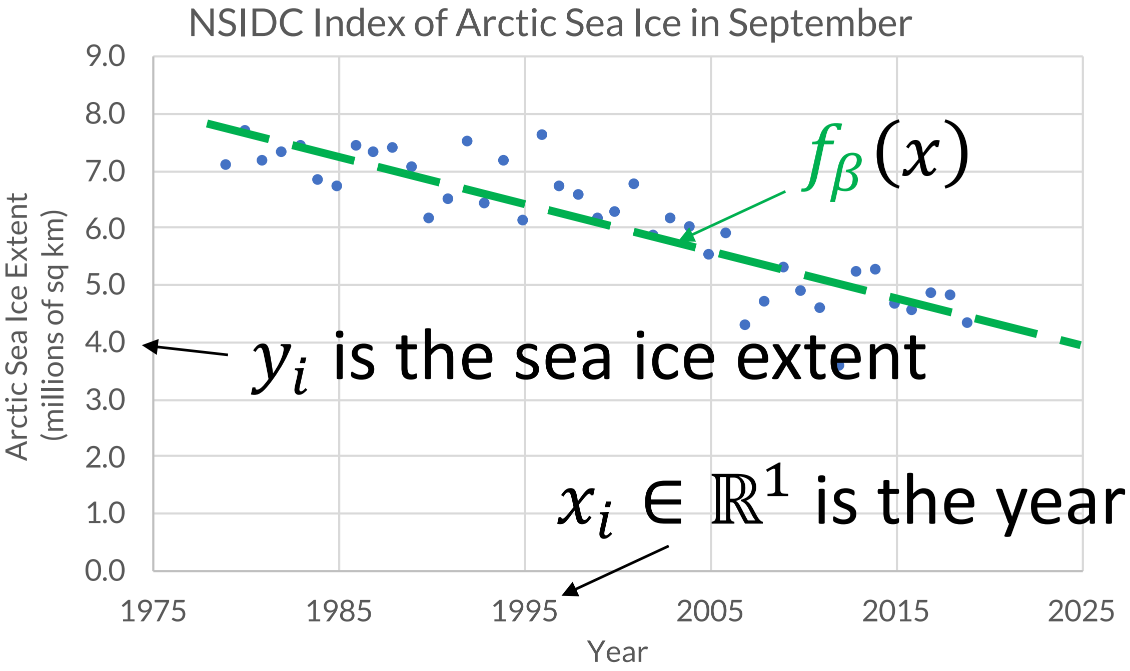


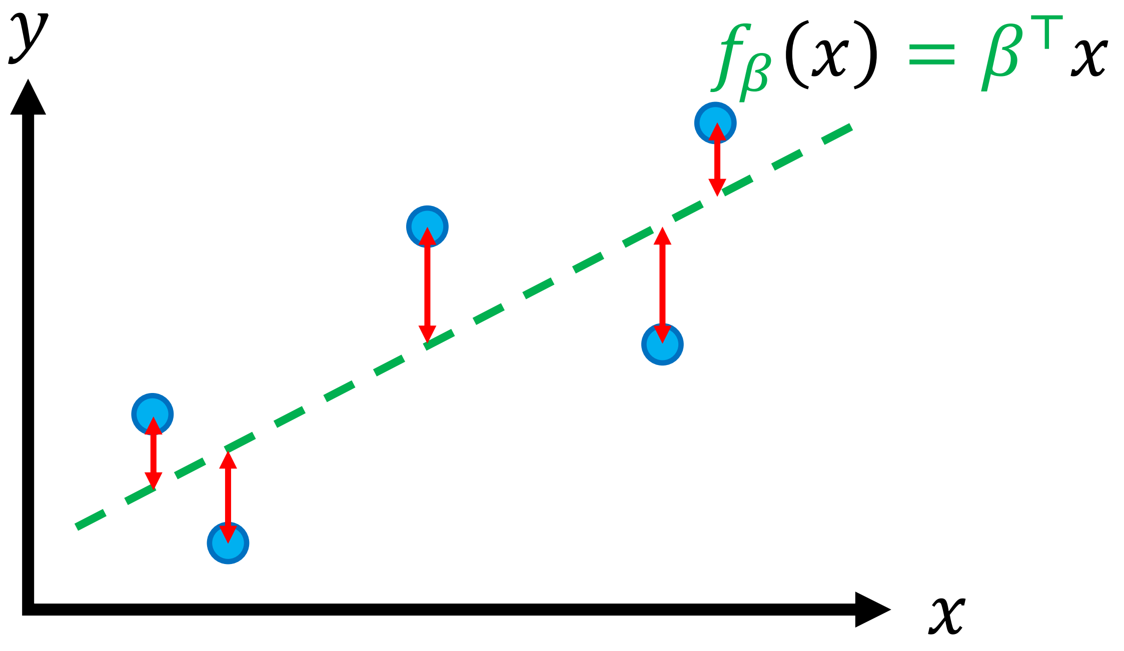
Image: https://www.flickr.com/photos/gsfc/5937599688/
Data from https://nsidc.org/arcticseaicenews/sea-ice-tools/

Choice of Loss Function

- For a single example,
  - $y_i \approx \beta^\top x_i$ if $(y_i - \beta^\top x_i)^2$ small
- **Mean squared error (MSE):**

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^\top x_i)^2$$

- Computationally convenient and works well in practice



$$L(\beta; Z) = \frac{\updownarrow^2 + \updownarrow^2 + \updownarrow^2 + \updownarrow^2 + \updownarrow^2}{n}$$

Linear Regression Problem, More Precisely

- **Input:** Data $Z = \{(x_1, y_1), \dots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$
- **Output:** A linear function $f_\beta(x) = \beta^\top x$ such that $y_i \approx \beta^\top x_i$

Linear Regression Problem, More Precisely

- **Input:** Data $Z = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$
- **Output:** A linear function $f_\beta(x) = \beta^\top x$ that minimizes the MSE:

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^\top x_i)^2$$

With these choices, the linear regression problem is sometimes called "Ordinary Least Squares" (OLS).

Linear Regression Algorithm

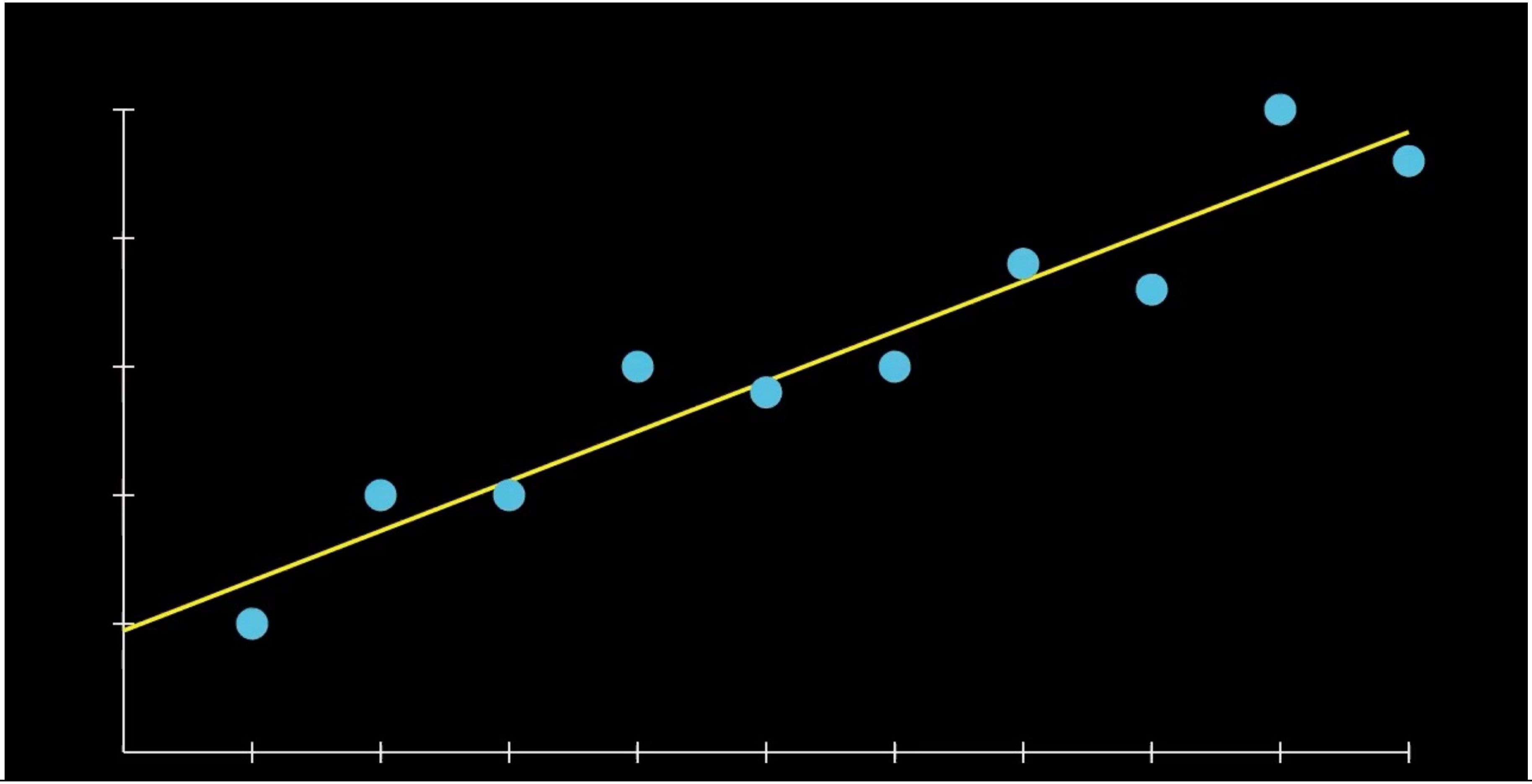
- **Input:** Dataset $Z = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- Compute

$$\hat{\beta}(Z) = \arg\min_{\beta \in \mathbb{R}^d} L(\beta; Z)$$
$$= \arg\min_{\beta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^\top x_i)^2$$

- **Output:** $f_{\hat{\beta}(Z)}(x) = \hat{\beta}(Z)^\top x$

We will later discuss how to find the parameters β that minimize the MSE loss L

Minimizing the Mean Squared Error

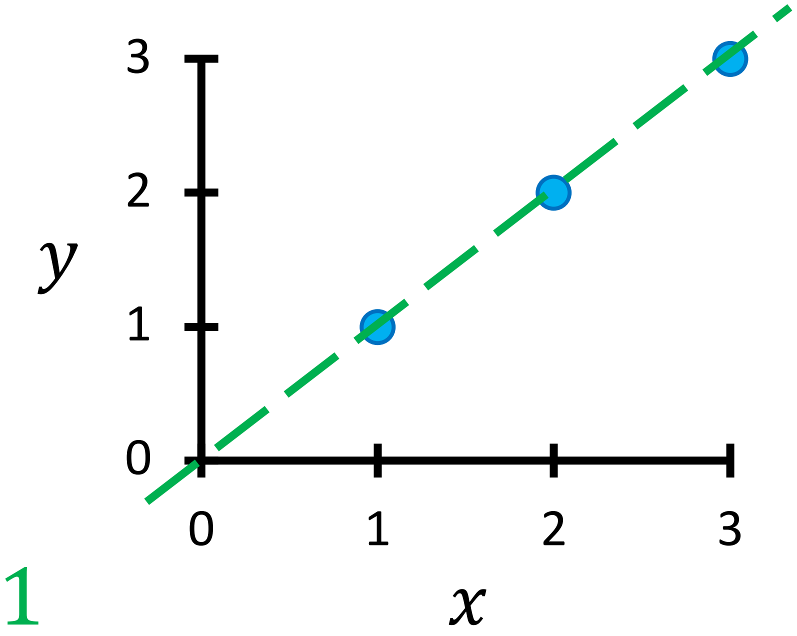


Q: What is depicted here is actually the “sum” of squared errors (SSE), but it doesn’t really matter. Why?

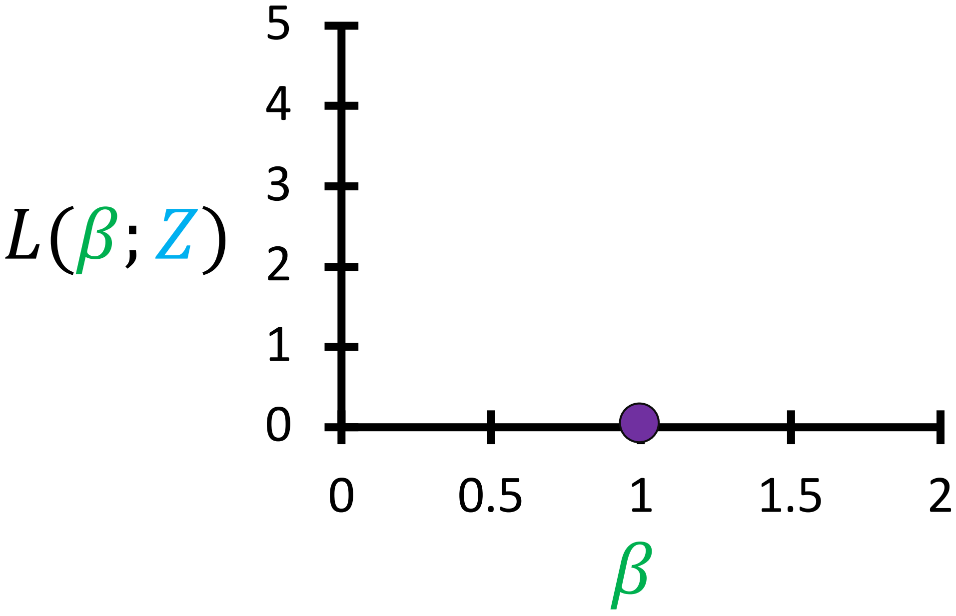
Youtube: 3-Minute Data Science

Intuition on Minimizing MSE Loss with 1-D inputs

- Consider $x \in \mathbb{R}$ and $\beta \in \mathbb{R}$, for the hypothesis class $y = \beta x$
- Then, $\mathrm{MSE} = L(\beta; Z) = \frac{1}{\mathrm{n}} \sum_{i=1}^{n} (y_i - \beta x_i)^2$

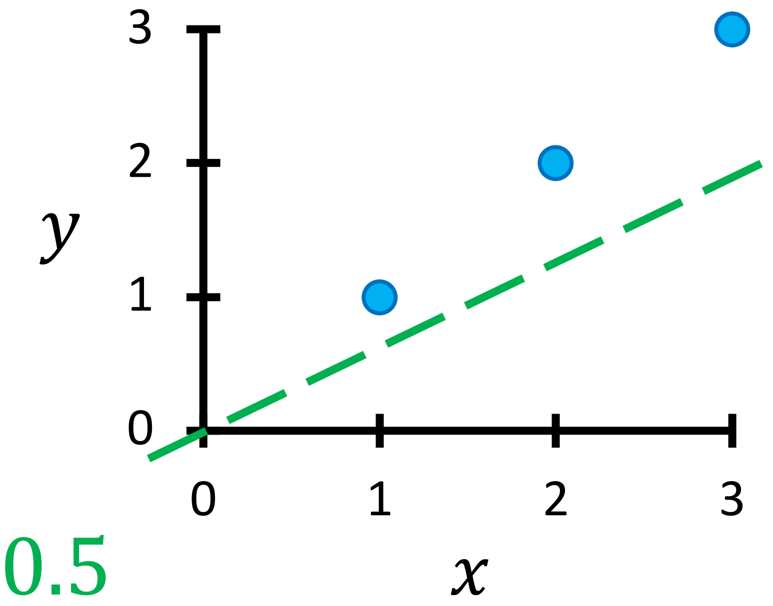


$\beta = 1$

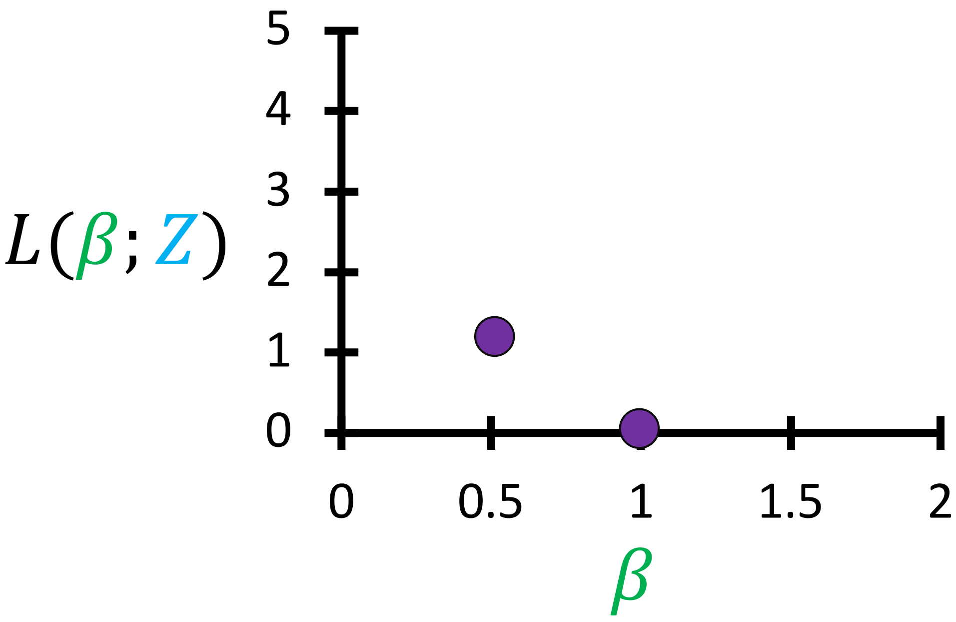


Intuition on Minimizing MSE Loss with 1-D inputs

- Consider $x \in \mathbb{R}$ and $\beta \in \mathbb{R}$, for the hypothesis class $y = \beta x$
- Then, $\text{MSE} = L(\beta; Z) = \frac{1}{\text{n}} \sum_{i=1}^{n} (y_i - \beta x_i)^2$

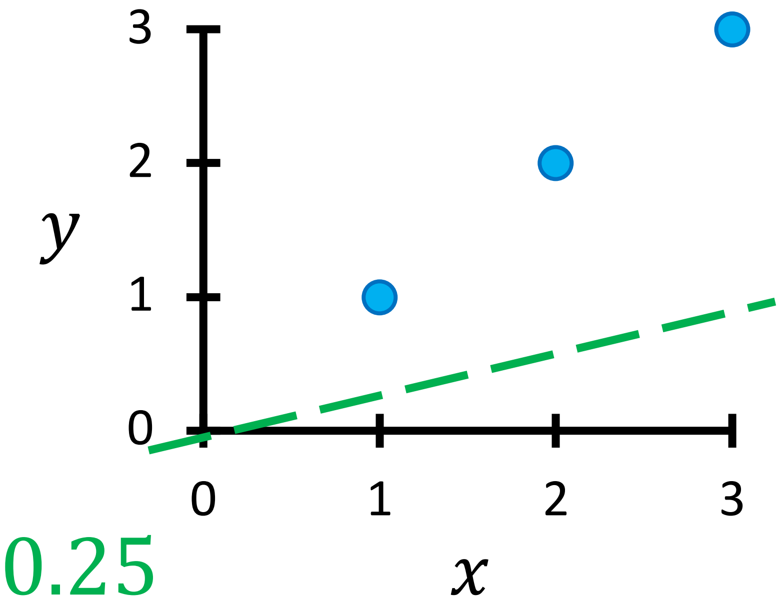


$\beta = 0.5$

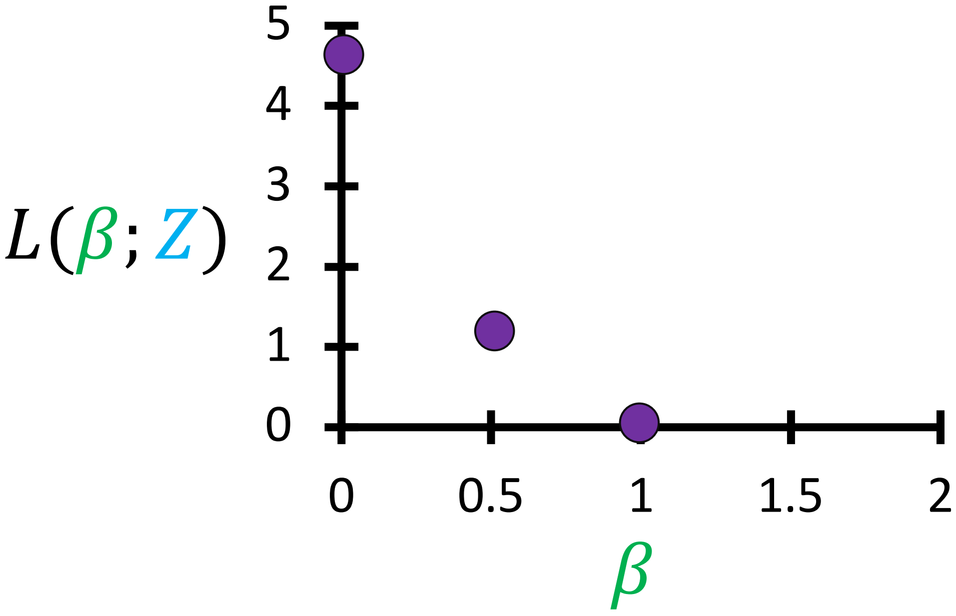


Intuition on Minimizing MSE Loss with 1-D inputs

- Consider $x \in \mathbb{R}$ and $\beta \in \mathbb{R}$, for the hypothesis class $y = \beta x$
- Then, $\mathrm{MSE} = L(\beta; Z) = \frac{1}{\mathrm{n}} \sum_{i=1}^{n} (y_i - \beta x_i)^2$

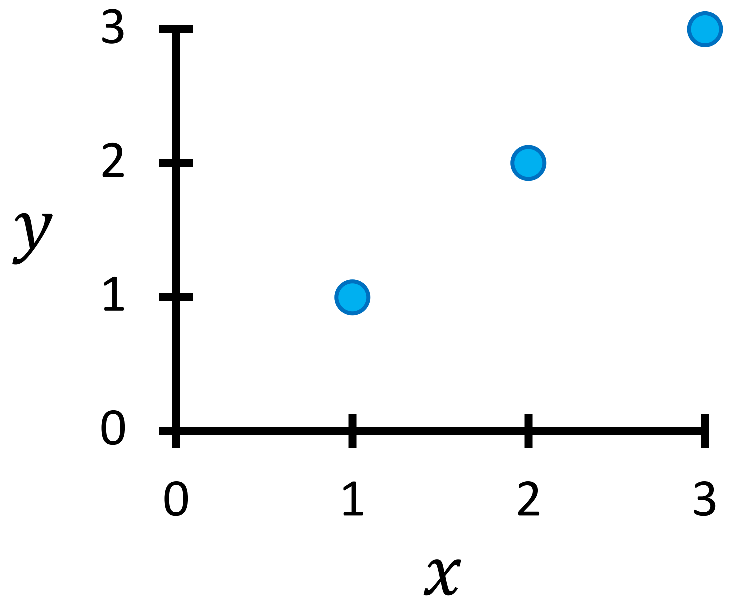


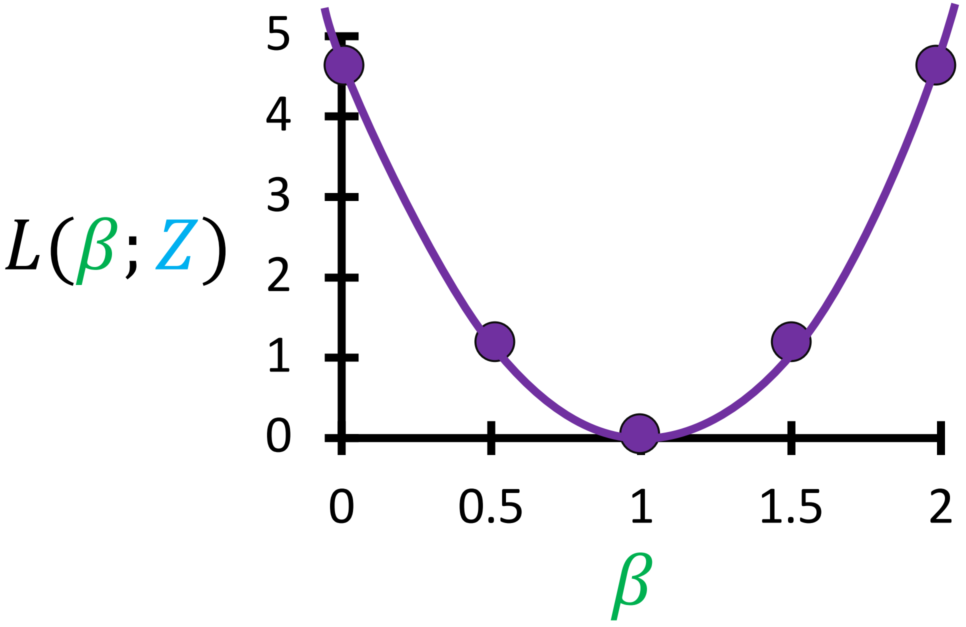
$\beta = 0.25$



Intuition on Minimizing MSE Loss with 1-D inputs

- Consider $x \in \mathbb{R}$ and $\beta \in \mathbb{R}$, for the hypothesis class $y = \beta x$
- Then, $\mathrm{MSE} = L(\beta; Z) = \frac{1}{\mathrm{n}} \sum_{i=1}^{n} (y_i - \beta x_i)^2$

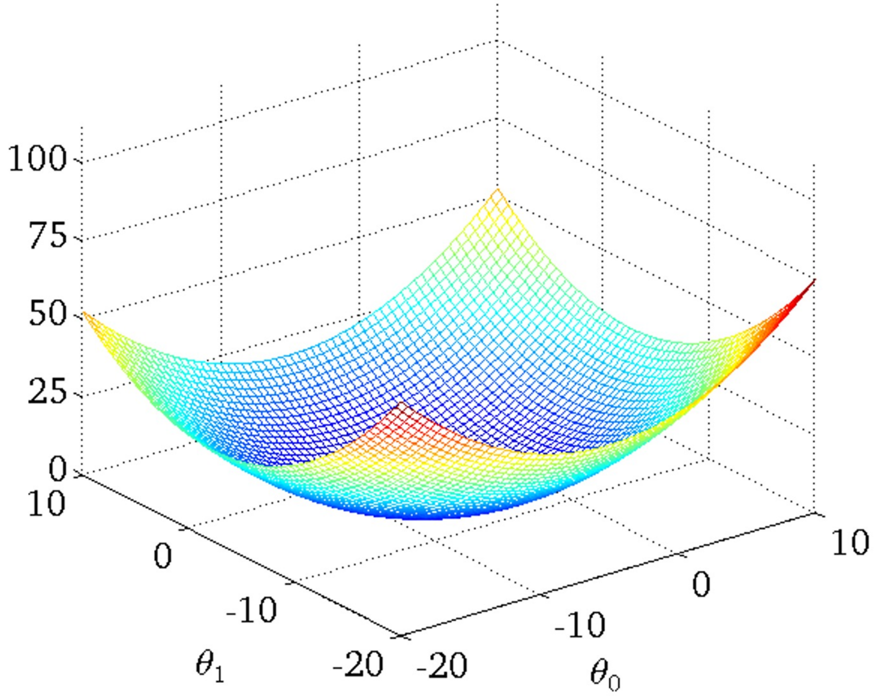


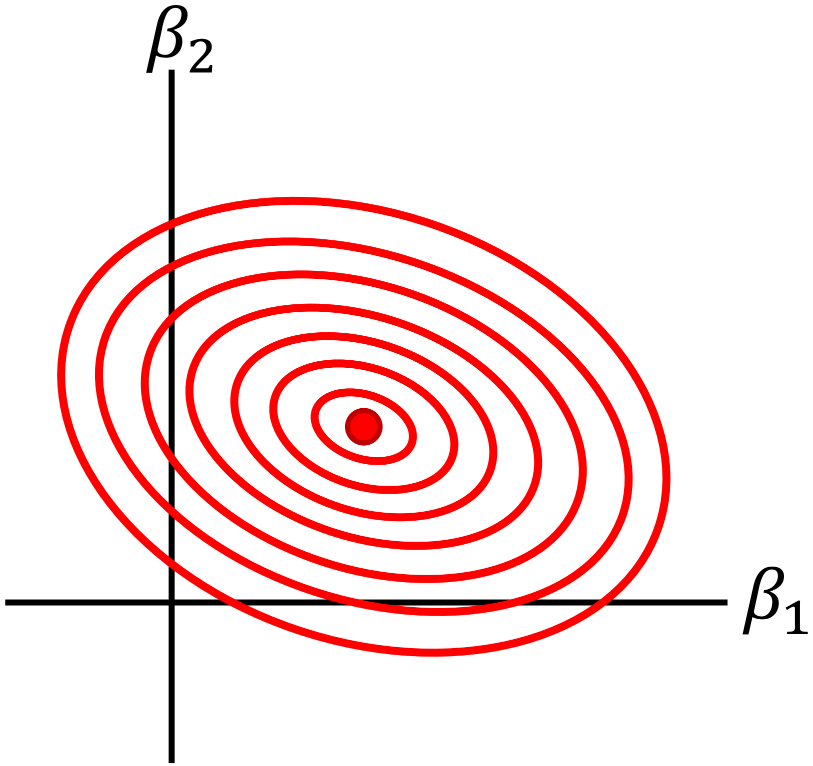


Intuition on Minimizing MSE Loss

- **Convex** ("bowl shaped") in general

$L(\beta; Z)$





Later, we will discuss how to find the parameters β that minimize the MSE loss L

What Is A “Good” Mean Squared Error?

- Zero MSE is rarely achievable. How do we know that the linear regression algorithm worked well?
- **Compare to simple baselines:** “Is my ML algorithm giving me more than what I could easily have coded up?” For example,
  - Constant prediction, e.g., predicting the mean of the training dataset target labels
  - Handcrafted model
  - …
- **A suite of performance metrics:** There’s no reason to solely rely on MSE for performance evaluation, even if you use MSE as the loss function.
- **Evaluate beyond the training examples**: (more on this soon)

Alternative Functions to Measure Performance

- **Mean absolute error:** $\frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$
- **Mean relative error:** $\frac{1}{n}\sum_{i=1}^{n} \frac{|\widehat{y_i} - y_i|}{|y_i|}$
- **$\boldsymbol{R^2}$ score:** $1 - \frac{\text{MSE}}{\text{Variance}}$
  - "Coefficient of determination"
  - Higher is better, $R^2 = 1$ is perfect

Alternative Functions to Measure Performance

- **Pearson correlation:** $\frac{1}{n}\sum_{i=1}^{n}\frac{(\hat{y}_i-\hat{\mu})(y_i-\mu)}{\hat{\sigma}\sigma}$
  - Usually estimated from some sampled measurements of those variables, and denoted as R (related to R^2 on the last slide!)

- **Rank-order correlation:**
  - First rank the measurements of $\hat{y}_{\boldsymbol{i}}$ and y separately, then replace each value in y by its rank, and ditto for $\hat{y}$
  - Then measure the linear correlation between those ranks

Performance Metrics

- Loss functions are special performance metrics.
  - Every loss function, e.g. MSE, is a performance metric, but not every performance metric is a convenient loss function for ML. (Reasons later)
- Always think carefully about the useful performance metric(s) for your ML problem. Use them to iterate on your ML design choices.
  - E.g. For an ML model that makes car driving decisions,
    - How frequently did it successfully get from A to B?
    - How fast did it get there?
    - How many traffic violations did it commit?
- The loss function is *a single scalar function*. A good choice of loss function:
  - expresses all the performance metrics.
  - is “convenient for machine learning.” More on this later.

Zooming Out of Linear Regression
To The Big Picture For a Bit …

Function Approximation View of ML

Data Z

Machine learning algorithm

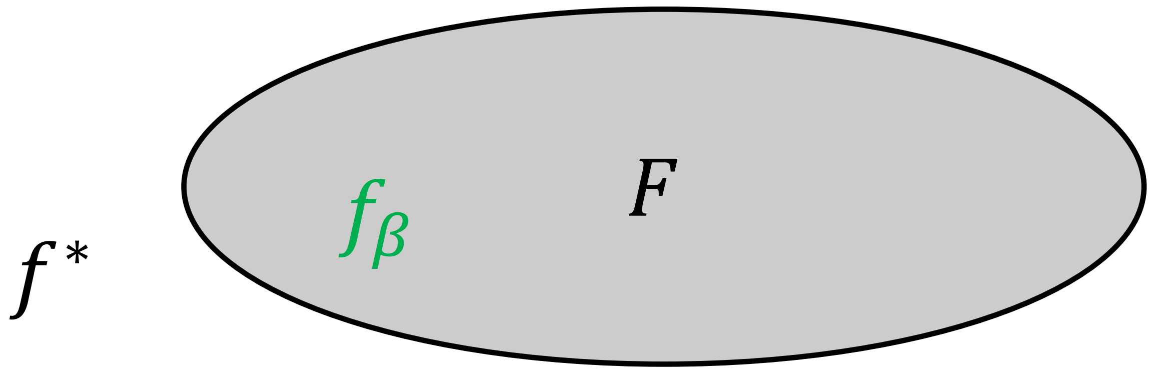
Model f

ML algorithm outputs a model f that best "approximates" the "true" function that generated data Z

The "True Function" f^*

- **Input:** Dataset Z
  - Presume there is an unknown function f^* that **generates** Z

- **Goal:** Find an approximation $f_\beta \approx f^*$ in our model family $f_\beta \in F$
  - Typically, f^* not in our model family F



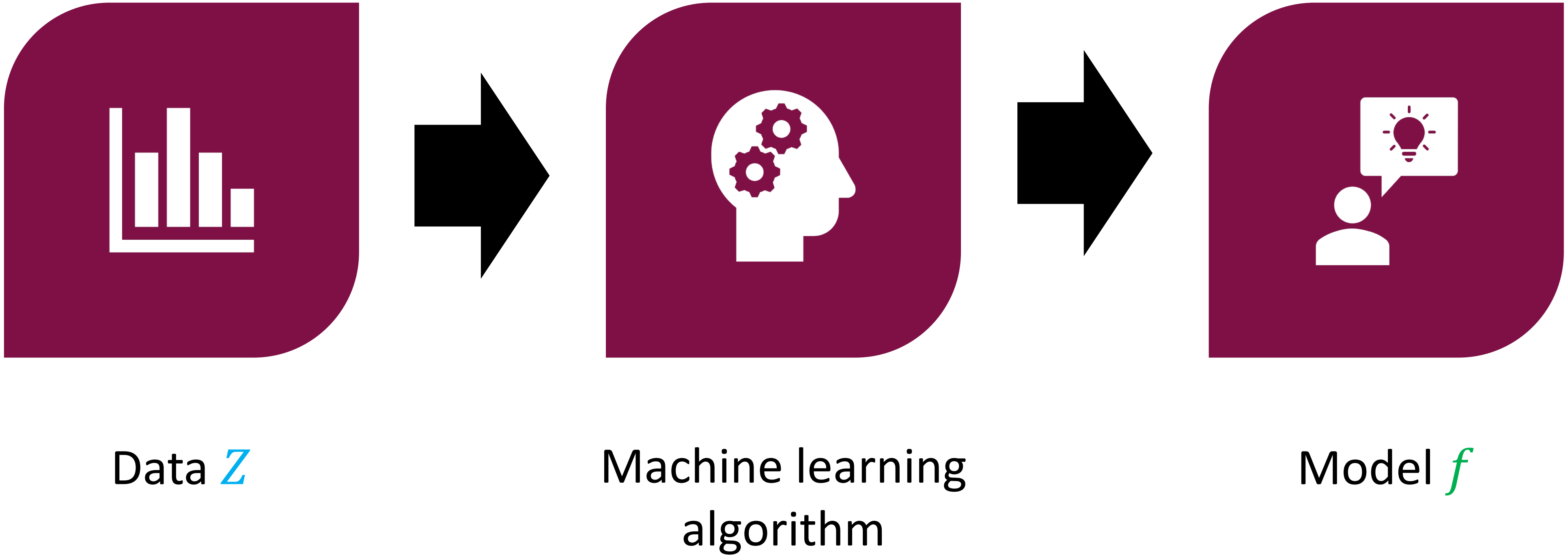
Function Approximation View of ML

- Framework for designing machine learning algorithms

- **Two key design decisions:**
  - What is the family of candidate models f?
  - How to define “approximating”?

Let us see how linear regression fits in this framework.

Machine Learning



Machine Learning as *Parametric Function Approximation*

Data Z

Machine learning algorithm

Model f_β

Parametric model family (i.e., $F = \{ f_\beta \mid \beta \in \mathbb{R}^d \}$)

Machine Learning as *Parametric Function Approximation*

Data Z

$\hat{\beta}(Z) = \arg\min_{\beta} L(\beta; Z)$

Model $f_{\hat{\beta}(Z)}$

ML algorithm minimizes loss of parameters β over data Z

… For *Supervised Learning*

Data Z $\qquad$ $\hat{\beta}(Z) = \arg\min_{\beta} L(\beta; Z)$ $\qquad$ Model $f_{\hat{\beta}(Z)}$

… For *Supervised Learning*

Data $Z = \{(x_i, y_i)\}_{i=1}^{n}$

$\hat{\beta}(Z) = \arg\min_{\beta} L(\beta; Z)$

L encodes $y_i \approx f_{\beta}(x_i)$

Model $f_{\hat{\beta}(Z)}$

Goal is for function to approximate **label** y given **input** x

… Specifically, *For Regression*

Data $Z = \{(x_i, y_i)\}_{i=1}^{n}$

$\hat{\beta}(Z) = \arg\min_{\beta} L(\beta; Z)$

L encodes $y_i \approx f_{\beta}(x_i)$

Model $f_{\hat{\beta}(Z)}$

Label is a real number $y_i \in \mathbb{R}$

… Specifically, *For Linear Regression*

Data $Z = \{(x_i, y_i)\}_{i=1}^{n}$

$\hat{\beta}(Z) = \arg\min_{\beta} L(\beta; Z)$

L encodes $y_i \approx f_{\beta}(x_i)$

Model $f_{\hat{\beta}(Z)}$

MSE loss

Model is a linear function $f_{\beta}(x) = \beta^{\top} x$

Linear Regression

General strategy

- Model family $F = \{f_\beta\}_\beta$
- Loss function $L(\beta; Z)$

Linear regression strategy

- Linear functions $F = \{f_\beta(x) = \beta^\top x\}$
- MSE $L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2$

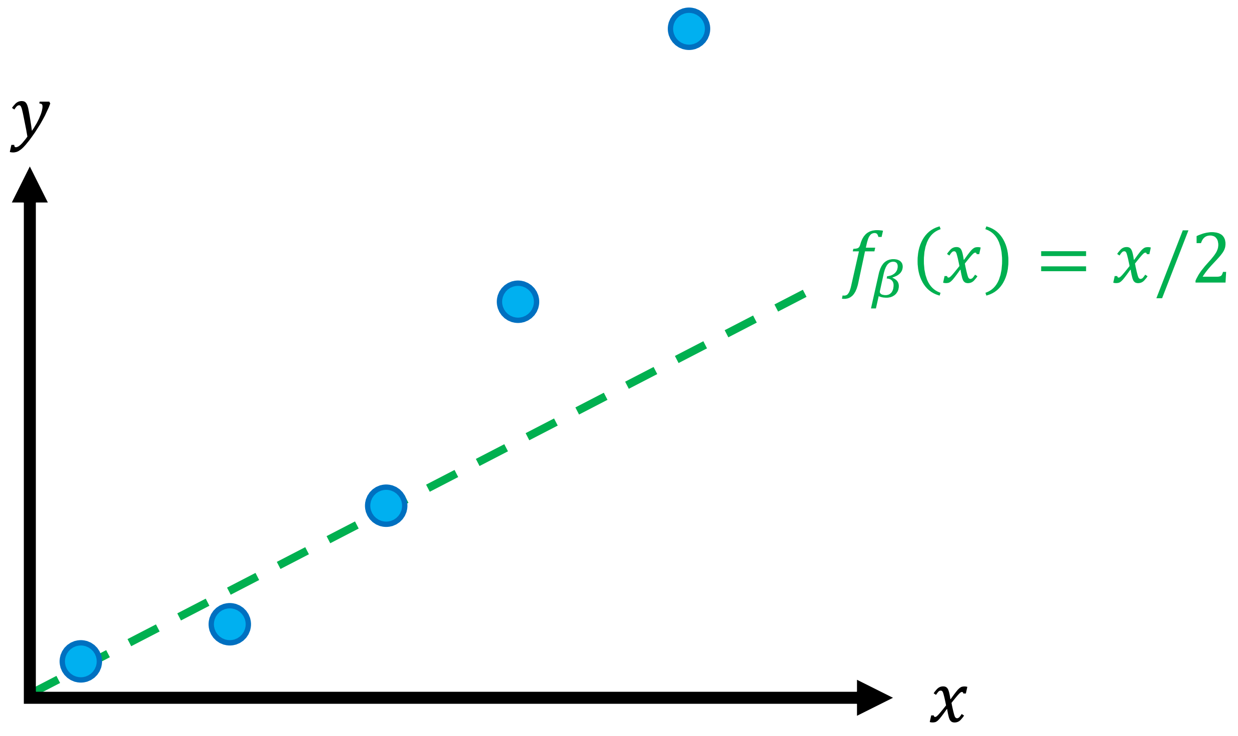
Linear regression algorithm

$$\hat{\beta}(Z) = \arg\min_{\beta} L(\beta; Z)$$

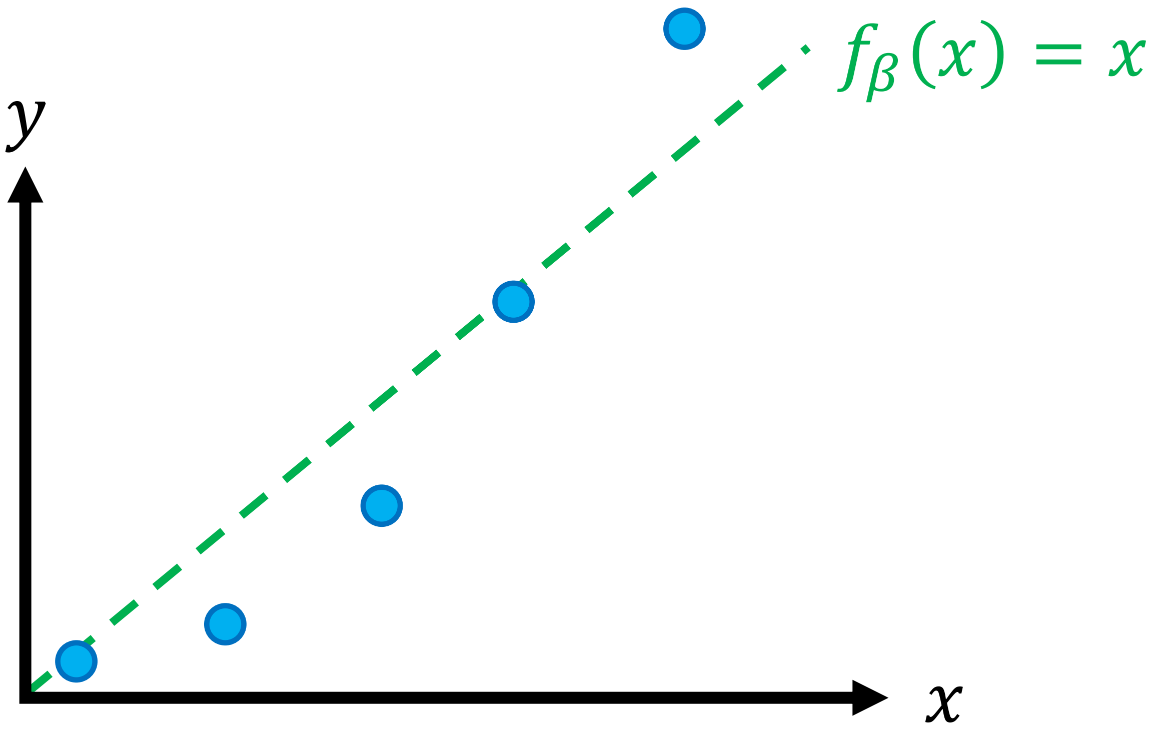
Linear Regression With Feature Maps

Linear Regression When Data is *Non-Linear?*

Example: Quadratic Function



Example: Quadratic Function



Can we get a better fit?

Feature Maps

General strategy

- Model family $F = \{f_\beta\}_\beta$
- Loss function $L(\beta; Z)$

Linear regression with feature map

- Linear functions over a given **feature map** $\phi: X \to \mathbb{R}^{d'}$

$$F = \{f_\beta(x) = \beta^\top \phi(x)\}$$

- MSE $L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta^\top \phi(x_i)\right)^2$

Quadratic Feature Map

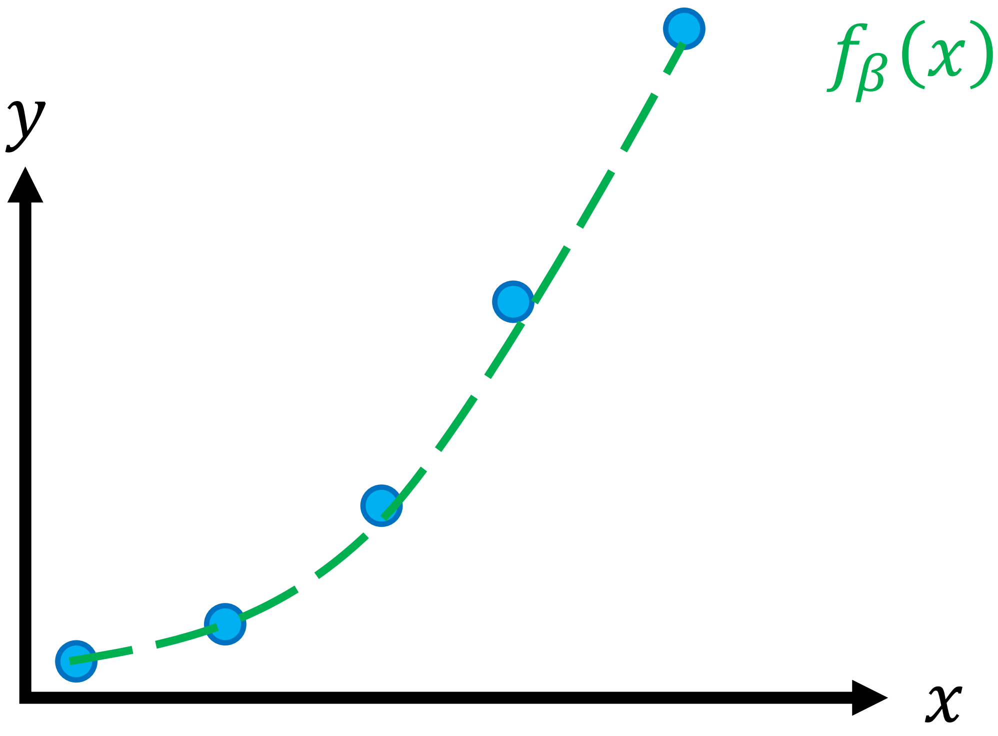
- Consider the feature map $\phi: \mathbb{R} \to \mathbb{R}^2$ given by

$$\phi(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix}$$

- Then, the model family is

$$f_\beta(x) = \beta_1 x + \beta_2 x^2$$

Quadratic Feature Map



In our family for $\beta = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$!

Feature Maps

- Effectively changes the hypothesis space! This is a powerful strategy for encoding “prior knowledge” about the function we are looking to approximate.

- **Terminology**
  - x is the **input** and $\phi(x)$ is the **features**
  - Often used interchangeably

Examples of Feature Maps

- Polynomial features
  - $\boldsymbol{\phi}(x) = [1, x_1, x_2, x_1^2, x_1 x_2, x_2^2]$
  - $f_\beta(x) = \beta_1 + \beta_2 x_1 + \beta_3 x_2 + \beta_4 x_1^2 + \beta_5 x_1 x_2 + \beta_6 x_2^2 + \cdots$
  - Quadratic features are very common; capture “feature interactions”
  - Can use other nonlinearities (exponential, logarithm, square root, etc.
- Note the intercept term (in red)
  - $\phi(x) = [1 \quad x_1 \quad \dots \quad x_d]^\top$
  - Almost always used; captures constant effect
- Encoding non-real inputs
  - E.g. Education level $x \in$ {“high school”, “college”, “masters”, “doctoral”}
    $\phi(x)$ maps to {1, 2, 3, 4}

Examples of Feature Maps

- Feature maps can also help handle very complex data like text and images
  - E.g., $x =$ "the food was good" and $y = 4$ stars
  - $\phi(x) = [1(\text{"good"} \in x) \quad 1(\text{"bad"} \in x) \quad \ldots]^\top$

- More on features for text and images later in the course!

Algorithm for Non-Linear Regression

First, select an appropriate feature map:

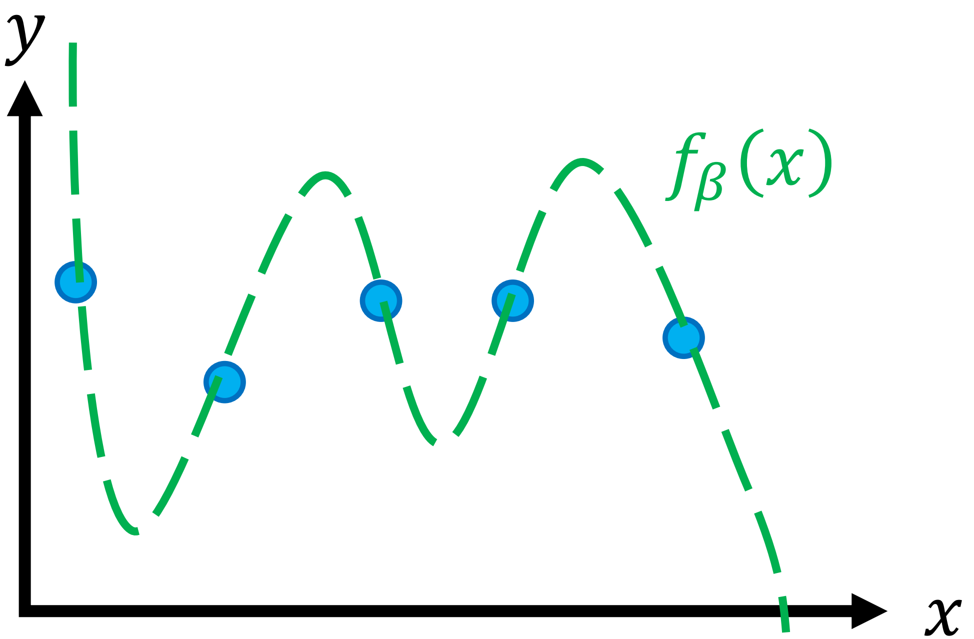
$$\boldsymbol{\phi}(x) = \begin{bmatrix} \phi_1(x) \\ \vdots \\ \phi_{d'}(x) \end{bmatrix}$$

Then, non-linear regression reduces to linear regression!

- Step 1: Compute $\boldsymbol{\phi}_i = \boldsymbol{\phi}(x_i)$ for each x_i in Z
- Step 2: Run linear regression with $Z' = \{(\boldsymbol{\phi}_1, y_1), \dots, (\boldsymbol{\phi}_n, y_n)\}$

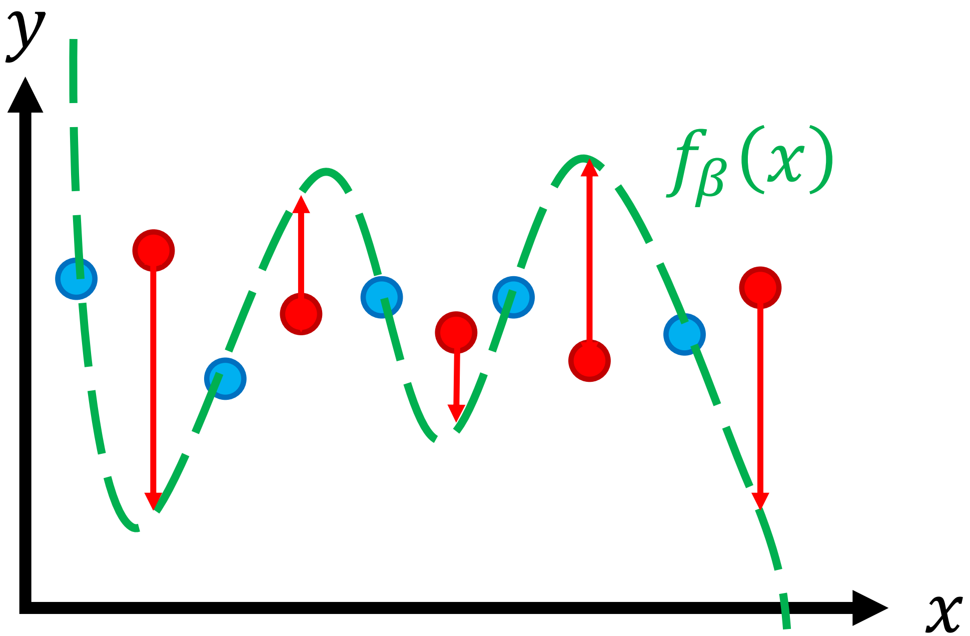
Question

- Why not always throw in lots of features?
  - After all, more features => more expressive hypothesis space!
  - For example, if $\boldsymbol{\phi}(x) = [1, x_1, x_2, x_1^2, x_1 x_2, x_2^2, \ldots]$
  - Can fit any n points using an n-th degree polynomial $f(x) = \beta_1 + \beta_2 x_1 + \beta_3 x_2 + \beta_4 x_1^2 + \beta_5 x_1 x_2 + \beta_6 x_2^2 + \cdots$



Generalization To Unseen Inputs

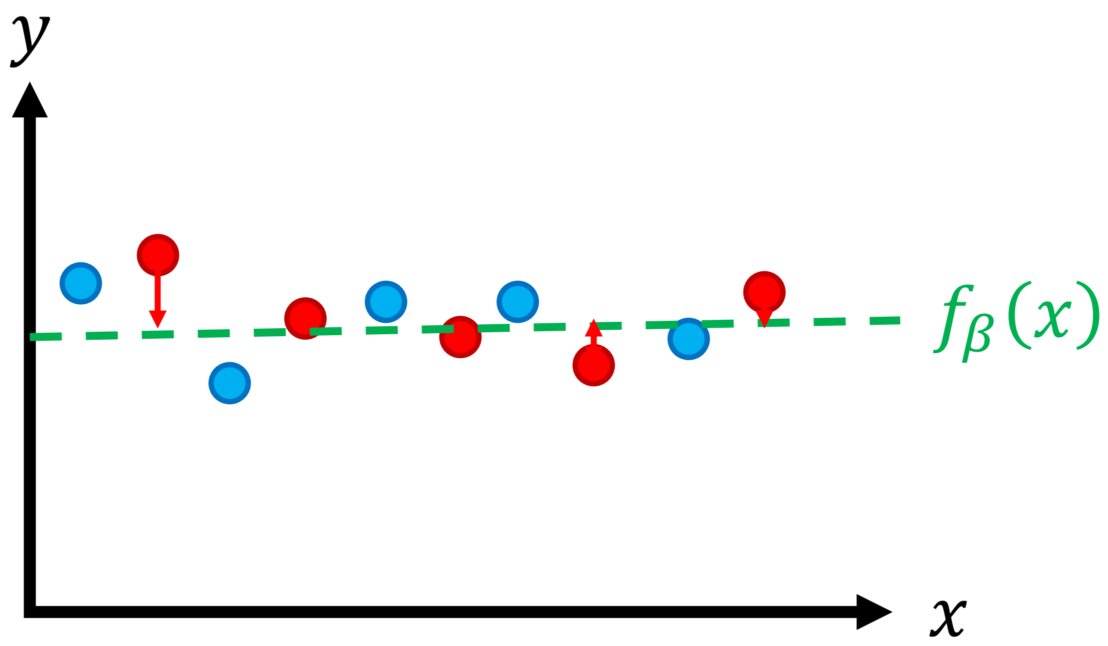
- **Issue:** The goal in machine learning is **generalization**
  - Given a **new** input x, predict the label $\hat{y} = f_\beta(x)$



The errors on new inputs are very large!

Generalization To Unseen Inputs

- **Issue:** The goal in machine learning is **generalization**
  - Given a **new** input x, predict the label $\hat{y} = f_\beta(x)$



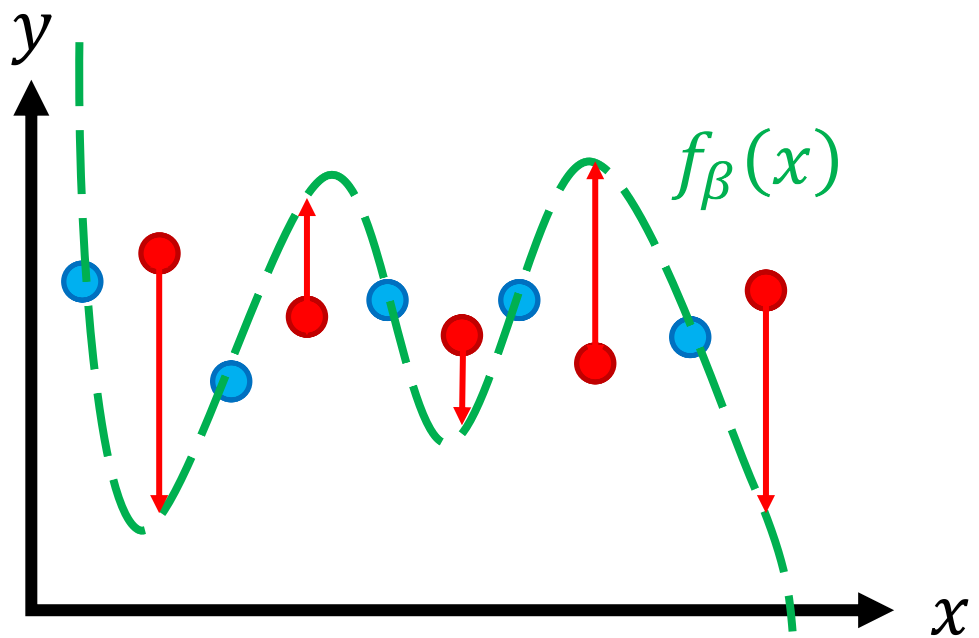
Vanilla linear regression actually works better!

Training vs. Test Data

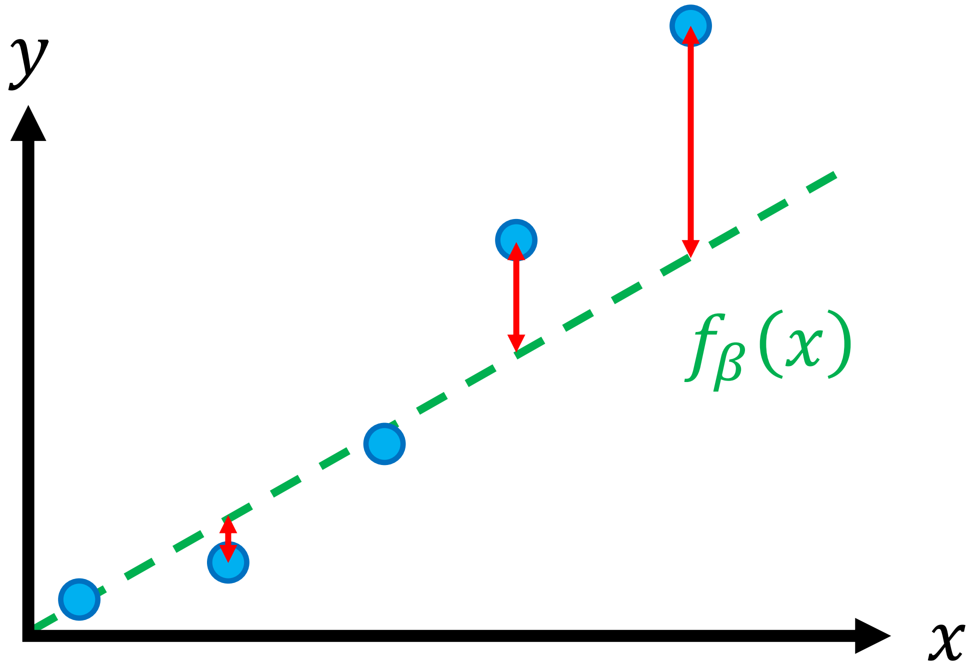
- **Training data:** Examples $Z = \{(x, y)\}$ used to fit our model
- **Test data:** New inputs x whose labels y we want to predict

Overfitting vs. Underfitting

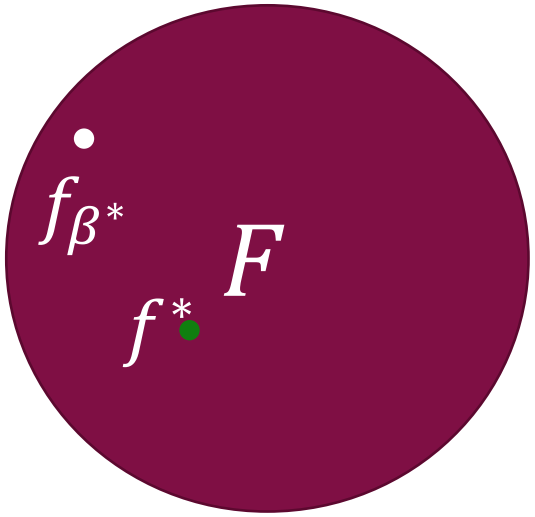
- **Overfitting**
  - Fit the **training data** Z well
  - Fit new **test data** (x, y) poorly

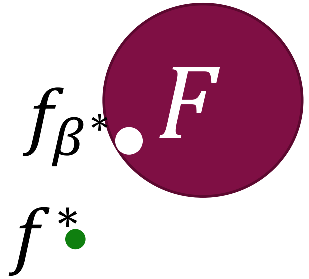


- **Underfitting**
  - Fit the **training data** Z poorly
  - (Necessarily also fit new **test data** (x, y) poorly)



Hypothesis Space, Overfitting, and Underfitting





Overfitting

Too many hypotheses in $\mathcal{H}$ that all fit the data well, Too little data, Noisy data

Underfitting

Inexpressive hypothesis space, i.e., no function in F that can approximate f^* on the data

“Noisy” Data

- **Noise in labels $\boldsymbol{y}_i$**
  - True data generating process is more complex than we can capture
  - May depend on unobserved features

- **Noise in features $\boldsymbol{x}_i$**
  - Measurement error in the feature values
  - Errors due to preprocessing
  - Some features might be irrelevant to the decision function